FLARE: Defending Federated Learning against Model Poisoning Attacks via Latent Space Representations

Ning Wang, *Member, IEEE*, Chaoyu Zhang, *Student Member, IEEE*, Yang Xiao, *Member, IEEE*, Yimin Chen, *Member, IEEE*, Wenjing Lou, *Fellow, IEEE*, and Y. Thomas Hou, *Fellow, IEEE*

Abstract—Federated learning (FL) has been shown vulnerable to a new class of adversarial attacks, known as *model poisoning attacks (MPA)*, where one or more malicious clients try to poison the global model by sending carefully crafted local model updates to the central parameter server. Existing defenses that have been fixated on analyzing model parameters show limited effectiveness in detecting such malicious models. In this work, we propose FLARE, a robust model aggregation mechanism for FL, which is resilient against state-of-the-art MPAs. Instead of solely depending on model parameters, FLARE leverages the *penultimate layer representations (PLRs)* of the model for characterizing the adversarial influence on each local model update. We further propose a trust evaluation method that estimates a trust score for each model update based on pairwise PLR discrepancies among all model updates. Under the assumption of honest majority, FLARE assigns a low trust score to model updates that are far from the benign cluster. FLARE then aggregates the model updates weighted by their trust scores and finally updates the global model. Extensive experimental results demonstrate the effectiveness of FLARE in defending FL against various MPAs, including semantic backdoor attacks, trojan backdoor attacks, and untargeted attacks, in various FL systems.

Index Terms—Poisoning attack, federated learning, trust score

1 Introduction

Machine learning (ML) is changing the ways people live and do business in every sector of our society. The success of ML, especially deep learning (DL), relies on the availability of powerful computers and massive amount of training data. However, learning systems that require all the data to be fed into a learning model running on a central server pose serious privacy concerns. For example, the transmission of health data across certain organizational boundaries may violate security and privacy rules such as those imposed by the Health Insurance Portability and Accountability Act (HIPAA[1]). Federated learning (FL) [2], [3], [4], which enables a group of intelligent agents to jointly learn a model while keeping their private data at their local devices, emerges as a promising new learning framework to address client data privacy problems.

FL has been applied in many popular applications, such as next-word prediction on Android Gboard by Google [5] and credit risk control by WeBank [6]. In an FL system, a large number of distributed clients cooperatively contribute to the learning process by uploading the gradients of their local models (or model weights) to the *parameter server* (*PS*) through multiple iterations without sharing the raw data at the clients. At the beginning of an FL task, *PS* initializes a global model. In each learning iteration, *PS* distributes the current global model parameters to selected clients. Each selected client continues to train the received model with its local data independently by following a predefined learning protocol. At the end of each learning iteration, *PS* collects and aggregates updates from clients using a gradient aggregation rule such as `FedAvg` [3]. *PS* then updates its global model and after multiple iterations *PS* outputs the final global model.

Despite many salient features of FL and its tremendous success in many applications, it has been shown recently that FL is vulnerable to model poisoning attacks (MPAs) [7], [8], [9], [10], [11]. In an MPA, the attacker (i.e., a malicious client) manipulates or crafts its model parameters sent to the *PS* in the aim of corrupting the global model by either increasing the prediction error (untargeted attacks) [10] or controlling the prediction on targeted inputs (backdoor attacks) [11], [12]. It is shown in [11] that even a single malicious attacker could deteriorate the global model accuracy and succeed in controlling the model output on chosen input data.

A potential countermeasure to MPAs is Byzantine resilient aggregation rules (BRARs) [13], [14], [15], which enable *PS* to learn an accurate global model when a bounded number of clients are malicious (i.e., Byzantine). Compared to straightforward aggregation rules that linearly combine the model updates (e.g., FedAvg [3]), BRARs (e.g., `Krum`

- *N. Wang is with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA (E-mail: ningw@usf.edu);*
- *C. Zhang and W. Lou are with the Department of Computer Science, Virginia Tech, Arlington, VA 22203, USA (E-mail: {chaoyu, wjlou}@vt.edu);*
- *Y. Xiao is with the Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA (E-mail: xiaoy@uky.edu);*
- *Y. Chen is with the Miner School of Computer and Information Sciences, University of Massachusetts Lowell, Lowell, MA 01854, USA (E-mail: ian_chen@uml.edu);*
- *Y.T. Hou is with the Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA (E-mail: thou@vt.edu)*

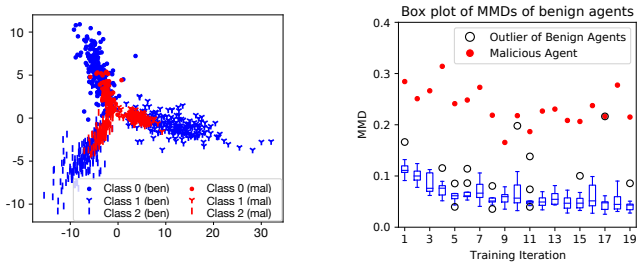
A preliminary version of this paper was published in ACM ASIACCS 2022 [1].

1. https://www.hhs.gov/hipaa/index.html

[13]) seek to provide statistical methods that are not abused by Byzantine values. To this end, BRARs leverage outlier-robust measures [16], e.g., median, trimmed estimator, to compute the center of updates despite the presence of Byzantine updates. Another line of defenses [17], [18] resorts to use anomaly detection methods to detect malicious local model updates and excludes them from the aggregation. MPAs have seen an increase in stealthiness and sophistication. The state-of-the-art MPAs [10], [11] can craft malicious model updates very similar to benign ones, breaking existing BRARs. Both the BRARs and ML-based defenses explore the model parameter space for detecting anomalous updates; they nonetheless show limited effectiveness in defending against the state-of-the-art MPAs [10], [11]. Many ML-based defenses [17], [18], [19] also need to collect a dataset of labeled benign and malicious models beforehand.

In this paper, we tackle the MPA challenge of FL through a new angle—the latent space representation of a model. We first make an important observation that even though the poisoned model parameters are very close to those of benign models, their representations in the latent space, provided an auxiliary input dataset, tend to diverge from those of benign models. Specifically, we target the *penultimate layer representation (PLR)* vector in the latent space and plot the PLRs of both attack-free models and poisoned models in Fig. 1a. It shows that the clean/benign PLRs follow the same distribution while the poisoned/malicious PLRs follow a different one. We made such observation consistently across different datasets and different neural network architectures. Besides the visual differences, to obtain quantifiable discrepancy, FLARE measures the distance (i.e., maximum mean discrepancy (MMD) [20]) between the PLRs of any two models. The average MMD scores of both poisoned models and clean models are illustrated in Fig. 1b, which confirms that PLR is a highly differentiating feature for poisonous models.

Based on the above observation, we propose FLARE (**F**ederated learning+**LA**tent-space **RE**presentations) to protect FL systems against state-of-the-art MPAs. FLARE features a novel methodology to estimate the trust score of a local model update by exploiting the similarity between its PLR to the PLR of others. Compared to defenses that only look into the model parameters, the PLR-based trust estimation enables FLARE to prevail in defending against carefully crafted malicious model updates. To estimate the trust score, FLARE computes a PLR sequence for each local model, which takes a very small auxiliary data at the *PS*. FLARE then exploits PLRs to distinguish malicious model updates from benign ones. Under the assumption that malicious clients are fewer than honest clients, FLARE assigns a trust score to each model update based on the pairwise PLR discrepancies among all model updates, in that those farther from the benign distribution are assigned lower scores. Finally, we employ a soft decision regime that aggregates model updates weighted by their trust scores. It is worth noting that FLARE performs trust score estimation based on the most recently received model parameters in each federated learning iteration, and it does not require collecting a dataset of model parameters beforehand, which yields efficiency advantage compared to existing ML-based defenses [17], [18], [19].



(a) 3-class clusters (b) MMD distribution

Fig. 1. A motivating example for our PLR approach using the fMNIST dataset [21]. (a) PLRs of 3 classes projected in a 2D space: benign models' PLRs are in blue and poisoned models' PLRs in red. (b) Averaged maximum mean discrepancy (MMD) between a model's PLRs and other models' PLRs. Box is for benign models, while red dots denote malicious model.

Contributions of this paper are summarized as follows:

- We propose FLARE, a novel detection and aggregation algorithm for FL to defend against state-of-the-art MPAs. Based on the key observation that PLRs of poisoned models tend to diverge from those of benign models, FLARE utilizes PLR for evaluating the trust score of a model update in FL. Based on the MMD of different local models' PLRs, FLARE features a trust estimation mechanism that assigns a trust score to each client and minimizes the impact of MPAs.
- Through theoretical analysis, we provide an Euclidean-distance-based interpretation on PLRs of deep neural network (DNN), justifying PLR as a promising measure to estimate the trust score of a model update.
- We add a case study on a real-time object detection system–you only look once. In addition to PLR, we evaluated the effectiveness of other deep inner-layer representations in detecting MPA, empirically demonstrating that deep inner-layer representation contains critical information for detecting MPA.
- Extensive experimental results demonstrate the effectiveness of FLARE for defending against state-of-the-art MPAs. FLARE outperforms existing defenses in terms of decreasing the attack success rate of MPAs. FLARE achieves consistent performance across various attack methods, and datasets, demonstrating the generality of the approach.

The subsequent sections of this paper are structured as follows: Section 2 delves into the background and related work. In Section 3, we present the system model and threat model. Section 4 offers a theoretical analysis of the penultimate layer representation. In Section 5, we elaborate on the proposed detection mechanism and system design. Sections 6 and 7 are dedicated to the discussion of the experimental setting and evaluation results, respectively. Section 8 presents a case study on YOLO, while Section 9 offers the conclusion.

2 Background and Related Work

Federated learning (FL), in a nutshell, allows a group of distributed clients to contribute their locally computed model parameter updates to the global model at the parameter server. The parameter server is responsible for distributing the initial model, collecting model parameter updates from agents, aggregating them through a certain aggregation rule, and adding the result to the global model. Eyeing on this FL paradigm, a class of stealthy attacks named model poisoning attacks (MPAs) have been demonstrated to be a significant threat to the security of FL systems [10], [11], [12], [22]. In an MPA, a compromised local agent attempts to corrupt the training process of FL by providing the parameter server carefully manipulated model parameters in each training iteration, in the aim of gradually degrading the FL model efficacy without being detected.

To protect the global model from malicious local updates in FL systems, BRARs were proposed in the literature, exampled by `Krum` [13], `Coomed`, `Trimmed Mean` [14], and `Bulyan` [15]. BRARs tackle the Byzantine attack/failure scenario in FL where a client does not follow the predefined learning protocol and sends arbitrary model updates to the *PS*. Technically, BRARs can bound the gap between the aggregated gradient and the true mean (i.e., without Byzantine clients) to a small value. Based on this feature, BRARs can partially address the MPA threat by preventing or downgrading the impact of some malicious model updates. Below we briefly introduce four state-of-the-art BRARs. `Krum` [13] selects one of n received updates $\{\delta_1, ..., \delta_n\}$ whose distance to the all the remaining update is the smallest. `Coomed` [14] selects the coordinate-wise median of n received updates as the final result. `Trimmed Mean` [14] first excludes the largest k values and the smallest k values in each coordinate. Then it calculates the average value of the remaining $(n - 2k)$ items. `Bulyan` [15] is a combination of `Krum` and `Trimmed Mean`. `Bulyan` firstly recursively applies `Krum` to select $(n - 2k)$ updates out of the total n updates. Then it applies `Trimmed Mean` to the selected $(n - 2k)$ updates to obtain the final result. We will use these BRARs for comparative analysis and evaluation.

Besides BRARs, a number of anomaly detection mechanisms are proposed to detect malicious local model updates. Shen et al. [18] proposed *Auror* to protect FL from malicious updates by filtering out-of-distribution parameters from the received model parameters; Fung et al. [19] proposed *FoolsGold* to identify poisoning Sybils based on the model similarity of client updates. Li et al. [17] proposed a spectral-anomaly-detection-based framework that detects the abnormal model updates based on their low-dimensional embeddings. Zhao et al. [23] proposed PDGAN for detecting poisoned models. PDGAN reconstructs training data from model updates and audits the accuracy for each participant model by using the generated data, and removes clients with accuracy lower than a predefined threshold. [24] uses a set of validating clients to determine if the (global) model-update derived in that round has been subject to a poisoning injection. That is, clients validate the global model on their local data, and vote for accepting or rejecting the model through a feedback loop. [25] proves that majority vote mechanism with ensemble federated learning is secure against MPA. The most relevant work to ours is FLTrust [26]. FLTrust bootstraps a trust score for each client based on its directional deviation from server model update and computes the average of the local model updates weighted by their trust scores as a global model update.

In the meantime, MPAs have seen an increase in stealthiness and sophistication. The backdoor MPAs proposed by Bhagoji et al. [11] incorporate a penalty on the distance between the crafted model parameters and the benign model parameters into its optimization objective. Bagdasaryan et al. [12] developed a generic constrain-and-scale technique that incorporates the evasion of defenses into the attacker's loss function during training. Similar techniques have been adopted in later works [27], [28] to achieve evasion of defenses. Meanwhile, Fang et al. [10] proposed untargeted MPAs to degrade the overall accuracy of the FL system by deviating the crafted model parameters from the true gradient direction. These MPAs [10], [12] have demonstrated their capability in evading existing defenses, e.g., `Krum`, `Trimmed Mean`, *Auror* and *FoolsGold*. [12] shows that an attacker is able to craft a malicious model satisfying that the Euclidean distance between the crafted model and any benign model is comparable or even less than the Euclidean distance among different benign models. Moreover, this crafted model can still misclassify an input to a target label. This attack makes the defenses by exploring Euclidean distance of model parameters useless and leads us to reconsider the defense for the MPAs.

We observe that most of the malicious model detection mechanisms [17], [18], [19], BRARs [13], [14], [15], and client credibility aggregation mechanisms (e.g., FLTrust [26] and [29], [30]) build their defense by directly analyzing the model updates from agents in the model parameter space. We also observe that due to the high dimensionality of the FL model as well as the non-smooth loss function, two models that are seemingly close in the parameter space may have dramatically different loss function. These defenses are likely to make miss detection on a malicious model that is carefully crafted to be similar to benign models in the parameter space. Based on this key insight, we propose to detect malicious local models by analyzing the latent-space features of models.

3 System Model

3.1 Federated Learning with Trust Scores

We consider a typical FL network with one parameter server *PS* and n participating clients $\{C_i\}|_{i\in[n]}$ (we define $[n] \coloneqq \{1, 2, ..., n\}$). The definition of frequently used symbols are shown in Table 1. Each client manages a local model (e.g., a neural network). At *PS*, the model weights of C_i are $\mathbf{w}_i \in \mathcal{W} \subseteq \mathbb{R}^d$, wherein $\mathcal{W}$ is the parameter space and d is the presumed model dimensionality. The global model parameter is denoted by $\theta \in \mathcal{W}$. We denote the model update from C_i as $\delta_i = \mathbf{w}_i - \theta$. Moreover, *PS* maintains a vector of trust scores for all clients, denoted $\{S_i\}_{i\in[n]}$. During normal operation, an FL task executes in iterations with *PS* acting as the model distributor and aggregator at the cloud side.

Fig. 2 illustrates the FL system model. At the system onset, *PS* initializes θ. Then each training iteration works as

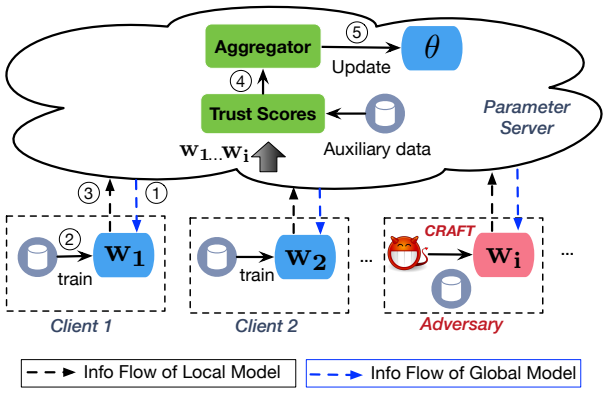


Fig. 2. Federated learning system model.

follows: (1) PS first selects multiple clients and distributes θ to them; (2) each of the selected clients, say C_i, initializes $\mathbf{w}_i = \theta$ and trains the model with its local data; (3) after the local training terminates, C_i provides its model update δ_i to PS; (4) PS uses the local model updates provided by the clients to compute a trust score S_i for every client C_i; (5) finally, PS aggregates local model updates weighted by their trust scores and updates the global model by: $\theta \leftarrow \theta + \sum_i S_i \delta_i$. At the end of the FL task, PS outputs the final global model.

We remark that the trust scores do not exist in the original FL formulation. As we show later, they allow our system FLARE to be responsive to malicious model updates before the aggregation step. We also assume PS has an auxiliary dataset $\mathcal{D} = \{x_i\}_{i \in [m]}$ containing a small number of records (e.g., $m = 10$), which will be used for PLR-based trust score evaluation in step (4). $\mathcal{D}$ can be obtained as long as we have one or more trusted clients in the FL system, who are willing to contribute to the system's security.

3.2 Threat Model

We assume the population of malicious clients is less than $0.5n$, in line with prior work [13], [14], [15], [26]. Meanwhile, every malicious client is a white-box adversary and can mount MPAs on the system, following from the state-of-the-art MPAs [10], [11].

White-Box Adversary. Being a valid FL client, the attacker has access to both the global model parameters and the model updates of other clients. Typically, the attacker estimates the model updates of other clients using a dummy model trained on its own clean data. Compared to a white-box attacker, a black-box attacker would only have access to the global model parameters. We opt for the challenging white-box adversary model to demonstrate the strength and effectiveness of our proposed defense.

Model Poisoning Attacks. According to the adversary goal, there are two types of MPAs: untargeted attacks and backdoor attacks. For an untargeted attack, the attacker aims to degrade the overall model accuracy. For a backdoor attack, the attacker aims to control the predictions on chosen input data records without degrading the overall prediction performance on other input data records. In specific, we use the two untargeted attacks in [10], which deviate the global model toward the opposite of the attack-free direction. We use the two backdoor attacks in [11], in which the adversary crafts malicious local models so as to inject a backdoor/trigger into the global model. The adversary maintains its stealth by decreasing the distance between the crafted model parameters and benign model parameters.

TABLE 1
Symbol definition.

Symbol	Definition
PS	parameter server
C_i	the i-th client
n	the number of clients
$\mathbf{w}_i$	the model parameters of C_i
θ	the global model parameters at PS
δ_i	the model parameter update of C_i
$\mathcal{D}$	the auxiliary dataset, $\lvert\mathcal{D}\rvert = m$
m	the number of data points in $\mathcal{D}$
c	the number of classes of the data
$\mathbf{x}$	the input image, $\mathbf{x} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$
$\mathbf{r}$	the penultimate layer representation, $\mathbf{r} \in \mathbb{R}^o$
$\mathbf{q}$	the confidence vector $\mathbf{q} \in \mathbb{R}^c$
R	a sequence of $\mathbf{r}$, $R = \{\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_m\}$
f	the mapping function of $\mathbf{x} \rightarrow \mathbf{q}$
g	the mapping function of $\mathbf{x} \rightarrow \mathbf{r}$
σ	the mapping function of $\mathbf{r} \rightarrow \mathbf{q}$
ω_i	model weights from the penultimate layer to the i-th output neuron
Ω	$\Omega = \{\omega_1, \omega_2, ..., \omega_c\}$
ct_i	the count that C_i is selected as others' nearest neighbor
S_i	the trust score of C_i

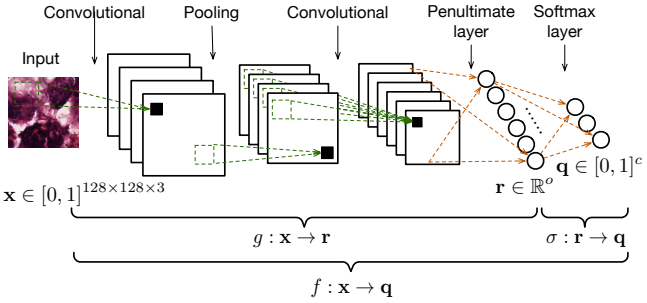


Fig. 3. The convolutional neural network architecture showing mapping functions f, g and σ.

4 Penultimate Layer Representation

We present the motivation, theory, and outlook for using penultimate layer representations (PLRs) to defend against MPAs.

4.1 PLR Basics

We use a convolutional neural network (CNN) for instance. Consider a CNN $f : \mathbb{R}^{d1 \times d2 \times d3} \rightarrow \mathbb{R}^c$, mapping points $x \in \mathbb{R}^{d1 \times d2 \times d3}$ to a c-dimensional probability vector $\mathbf{q} \in \mathbb{R}^c$, where c is the number of classes. We consider an image input and use $d1$, $d2$, and $d3$ to represent an image's width, height, and number of channels. Let the last layer of the network be a softmax layer. The mapping function from the input to the penultimate layer (i.e., the layer before the last layer) is denoted by $g : \mathbb{R}^{d1 \times d2 \times d3} \rightarrow \mathbb{R}^o$. The output of function g is a PLR which is denoted by $\mathbf{r} \in \mathbb{R}^o$. We use σ to denote the mapping function from PLR to the output probability vector, $\sigma : \mathbf{r} \in \mathbb{R}^o \rightarrow \mathbf{q} \in \mathbb{R}^c$. The mapping functions are shown in Fig. 3.

4.2 Power of PLR in Separating MPAs

Next we show that PLR exhibits highly differentiating power in detecting malicious models crafted by the advanced attacks. This is in contrast to solely looking at the model parameters.

The prediction of a c-class classifier on an input is a probability vector $\mathbf{q} = [q_1, q_2, ..., q_c]$, where q_k represents the likelihood the model assigns label k to the input and $\sum_{k=1}^{c} q_k = 1$. We use $\Omega = [\omega_1, , \omega_2, ..., \omega_c]$ to represent the weight connecting the penultimate layer to the last layer where $\omega_k \in \mathbb{R}^o$ denotes the weights connecting to the k-th neuron of the output (i.e., softmax) layer. According to the softmax function, q_k is calculated as

$$q_k = \frac{\exp(\mathbf{r}^T \omega_k)}{\sum_{i=1}^{c} \exp(\mathbf{r}^T \omega_i)}. \quad (1)$$

We interpret the output probability using the Euclidean distance between PLR and templates ω_k.

Proposition 1. In a c-class NN classifier where the last two layers are fully connected and the last layer is a softmax layer, the output probabilities of any two class k and l ($\forall k, l \in [c]$ and $k \neq l$) satisfy that $q_k > q_l$ if

$$\|\mathbf{r} - \omega_l\|_2 - \|\mathbf{r} - \omega_k\|_2 \geq C_{kl}, \quad (2)$$

where $\mathbf{r}$ represents the PLR of an input data record and ω_k is the weights connecting to the k-th neuron of the output layer. $||\mathbf{r} - \omega_k||_2$ denotes the Euclidean distance between $\mathbf{r}$ and template ω_k, i.e., $||\mathbf{r} - \omega_k||_2 = \mathbf{r}^T\mathbf{r} - 2\mathbf{r}^{\mathbf{T}}\omega_k + \omega_k^T\omega_k$. C_{kl} is a constant and $C_{kl} = \omega_l^T\omega_l - \omega_k^T\omega_k$.

Proposition 1 (see proof in Appendix) implies that the smaller the distance between $\mathbf{r}$ and the template ω_k (when the distance between $\mathbf{r}$ and other templates is fixed), the larger the likelihood that $\mathbf{r}$ is classified as class k. Here, we can regard the template ω_k as the cluster center of class k. Classification can be determined by comparing a target PLR to all the c templates. For each pair of classes, e.g., k and l, the input is more likely to be class k if Eq. (2) is satisfied or class l otherwise. Based on Proposition 1, we hypothesize that the PLRS of inputs belonging to one specific class exhibit a relatively small Euclidean distance to the corresponding template, possibly resulting in a consistent pattern (i.e., a cluster centered at the template). We then analyze how the distortion in PLR (i.e., $\|\mathbf{r}_1 - \mathbf{r}_2\|_2$) transforms to the final output probability vector.

Proposition 2. The mapping function $\sigma : \mathbf{r} \in \mathbb{R}^o \rightarrow \mathbf{q} \in \mathbb{R}^c$ maps a PLR to a probability vector as discussed above. For any two PLRs $\mathbf{r}_1, \mathbf{r}_2$, we have

$$\|\mathbf{q}^1 - \mathbf{q}^2\|_2 \leq \|\Omega\|_2 \|\mathbf{r}_1 - \mathbf{r}_2\|_2, \quad (3)$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ are the PLR of two input x_1 and x_2 respectively. $\mathbf{q}^1$ and $(\mathbf{q}^2)$ are the output probability vector for input x_1 and x_2 respectively.

Proposition 2 (see proof in Appendix) implies that a difference in the PLR space will transform to the difference of the corresponding output probability. The output probability will be very similar if $\|\mathbf{r}_1 - \mathbf{r}_2\|_2$ is small enough.

Furthermore, we consider two local models in FL. The weights Ω of the two local models should be very similar since the local models begin their training with the same initialization and only perform several training steps. The two models transform the same input x (belonging to class c) into PLR $\mathbf{r}_1$ and PLR $\mathbf{r}_2$ respectively. $\mathbf{r}_1$ and $\mathbf{r}_2$ should stay within a certain distance if the two models would give a similar prediction confidence vector on x. We further demonstrate our hypothesis by visualizing the PLRs of benign models in Fig. 4. Here we assume ten participating clients in an FL system, including nine benign clients and one attacker. The attacker manipulates its model parameters by implementing a well-known backdoor MPA [11]. We collect the ten local models and calculate a PLR for each local model using the same data point. We plot the ten versions of PLR in a 2-D space. Ten versions of PLR are presented in Fig. 4. We can see that all the benign PLRs stays close to each other while the malicious one exhibits a more significant distance from the benign ones. We show three examples (three input data points), and we get a similar observation among all three examples.

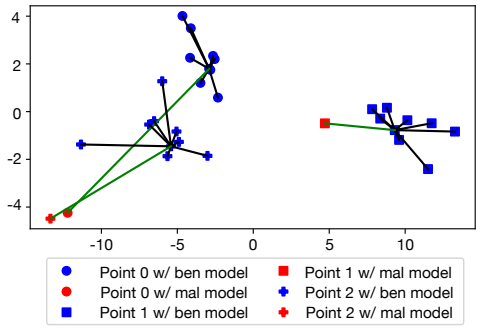


Fig. 4. PLRs and inter-distance between PLRs in the 2-D space of the Kather dataset [31]. There are three types of markers (i.e., circle, plus, and square), each type of maker corresponds to one input data points. There are ten items of each type of marker, representing ten versions of PLRs for one input data. Nine (in blue) are from nine benign local models and one (in red) is from a malicious model.

4.3 Visualizing PLRs Distribution

Based on the above theoretical analysis and empirical results, we find that the poisoned model produces a PLR that is relatively far from the cluster of benign PLRs. We further plot more data points from the same class to see the distribution of PLRs. In Fig. 6, we plot the PLRs of both malicious models and benign models under backdoor attacks. We observe that the PLRs of benign models follow a distribution while the PLRs of malicious models deviate from it. We present the results under untargeted attacks and achieve consistent result (see Fig. 10). All the results further confirm that PLR is a promising feature for detecting poisoned models. The key reason for distribution deviation is because the PLR distance among benign models is smaller than the PLR distance between benign models and malicious models.

The visualization procedure used to illustrate the PLRs is as follows [32]: 1) randomly select three classes from all classes; 2) compute the orthonormal basis of the hyperplane on which the templates of the selected class reside; 3) project the PLRs of data records from the three classes to the hyperplane (i.e., calculating the inner product of the PLRs and the

orthonormal basis); and 4) reduce the dimensionality of the results in Step 3) to 2-D space by applying PCA. Finally, we can plot the PLRs in 2-D space.

5 FLARE: Defending against MPAs

5.1 Overview of FLARE

Based on the observation and theoretical analysis of PLRs, we design our system—FLARE. The overview of FLARE is shown in Fig. 5. Our design is compatible with the general FL system. One practitioner can easily apply FLARE to an FL system by adding a trust score estimation module. As shown in Fig. 5, local clients firstly submit their local model updates δ_i to the *PS*. *PS* calculate PLRs using an auxiliary dataset for each local model. Next, FLARE computes the nearest neighbors of each local model based on the Maximum Mean Discrepancy (MMD) of PLRs. A client's count to be selected as other clients' nearest neighbor is used to estimate its trust score S_i. Finally, *PS* aggregates model updates weighted by their trust scores and use it to update the global model. The workflow of FLARE is also shown in Algorithm 1.

5.2 Detailed Design

FLARE features two differences at *PS* comparing to the traditional FL system: 1) At the very beginning, *PS* initializes the global model by training with the auxiliary dataset $\mathcal{D}$ instead of performing random initialization. This procedure can accelerate the training process. It also helps clients make correct predictions on $\mathcal{D}$, which increases the probability that the benign models' PLR follow one distribution. 2) At the aggregation stage of each learning iteration, *PS* estimate a trust score for every model update and aggregate them by their trust scores. Next we elaborate on FLARE's aggregation scheme, mainly how to estimate the trust scores.

Firstly, *PS* computes PLRs for each local model $\mathbf{w}_j$ using $\mathcal{D} = \{\mathbf{x}_i\}_{i\in[m]}$. The mapping function of the model with weight $\mathbf{w}_j$ is represented by $g_{\mathbf{w_j}}$: $\mathbf{x} \in \mathbb{R}^{d1\times d2\times d3} \rightarrow \mathbf{r} \in \mathbb{R}^o$ where $d1$, $d2$, and $d3$ represent the width, height, and channels of an image input and o represents the dimensionality of a PLR. As we have m data points in $\mathcal{D}$, we can get m PLRs $\{\mathbf{r}_1, ..., \mathbf{r}_m\}$ where $\mathbf{r}_i = g_{\mathbf{w_j}}(\mathbf{x_i})$. To distinguish between PLRs of different models, we use $R_j := \{g_{\mathbf{w_j}}(\mathbf{x_1}), ..., g_{\mathbf{w_j}}(\mathbf{x_m})\}$ to represent the PLRs by the j-th model.

Next, FLARE applies MMD [20] on R_i and R_j to test whether the two PLR sequences follow the same distribution. We choose MMD but not other two-sample test methods mainly because the number of PLR points $m = 10$ in one sample is much smaller than the dimensionality $o = 128$ of the data. It is difficult to measure the distribution of such a small sample. Traditional parametric two-sample test methods usually have strong assumptions about the parameters of the population distribution from which the sample is drawn and therefore are not applicable. FLARE utilizes MMD to estimate the distance between two PLR sequences since MMD does not require knowing the PLR distribution. Without loss of generality, the unbiased estimate of MMD between the two PLR sequences R_i and R_j is:

$$\text{MMD}(R_i, R_j) = \frac{1}{m(m-1)}\Bigg[\sum_{a\in R_i}\sum_{b\in R_i, b\neq a} k(\mathbf{a}, b) + \sum_{a\in R_j}\sum_{b\in R_j, b\neq a} k(a, b) - 2\sum_{a\in R_i}\sum_{b\in R_j} k(a, b)\Bigg] \tag{4}$$

where $k(\cdot)$ is a Gaussian kernel function. We expect the empirical test statistic $\text{MMD}(R_1, R_2)$ to be small if R_1 and R_2 are from an identical distribution, and large if the distributions are far apart. We use the shortcut $MMD_{ij} = \text{MMD}(R_i, R_j)$ to represent the MMD between the i-th model's PLRs and j-th model's PLRs.

FLARE utilizes the count of nearest neighbors to estimate the trust score of a model update. *PS* selects the top 50% nearest neighbors for each local model based on the MMD scores. The count ct_i for $\mathbf{w_i}$ increases by one once $\mathbf{w_i}$ is selected by any $\mathbf{w_j}(j \neq i)$. The count ct_i value indicates the trustworthiness degree. We then use the softmax function with temperature to transform the count value into a trust score:

$$S_i = \frac{\exp(ct_i/\tau)}{\sum_{k=1}^{n}\exp(ct_k/\tau)}, \tag{5}$$

where τ is the temperature parameter. S_i is in the interval of $[0, 1]$ and $\sum_i^n S_i = 1$. For a sequence of $\{ct_i\}_{i\in[n]}$, a larger τ will output more even trust scores. We can select a smaller τ to highlight benign model updates and reduce the weights of suspicious model updates. We use $\tau = 1$ in our paper.

An alternative scheme is to use the average MMD value of one model to other models to estimate its trust score. The reason why we select the nearest-neighbor-count-based scheme but not the average-MMD-based scheme is shown as follows. The nearest-neighbor-count-based scheme is more resilient to collusive attackers than the average-MMD-based scheme. Colluded attackers can produce nearly the same PLRs, thus resulting in extremely small MMD with each other. In this way, the final average MMD of an attacker may be smaller than benign models, making the detection scheme useless. On the other hand, the counts of the nearest neighbor can deal with this type of collusion when attackers are less than 50% of all clients.

Finally, we aggregate model updates weighted by their trust scores and use it to update the global model by

$$\theta \leftarrow \theta + \sum_{i=1}^{n} S_i\delta_i. \tag{6}$$

where n is the number of local model updates received by *PS*.

6 Implementation and Experimental Settings

We implement MPAs and FLARE on the TensorFlow platform. We run all the experiments on a server equipped with an Intel Core i7-8700K CPU 3.70GHz×12, a GeForce RTX 2080 Ti GPU, and Ubuntu 18.04.3 LTS. We implement the following four types of MPAs: `Attack-Krum-Untargeted`, `Attack-TM-Untargeted` [10], `Attack-Krum-Backdoor`, and

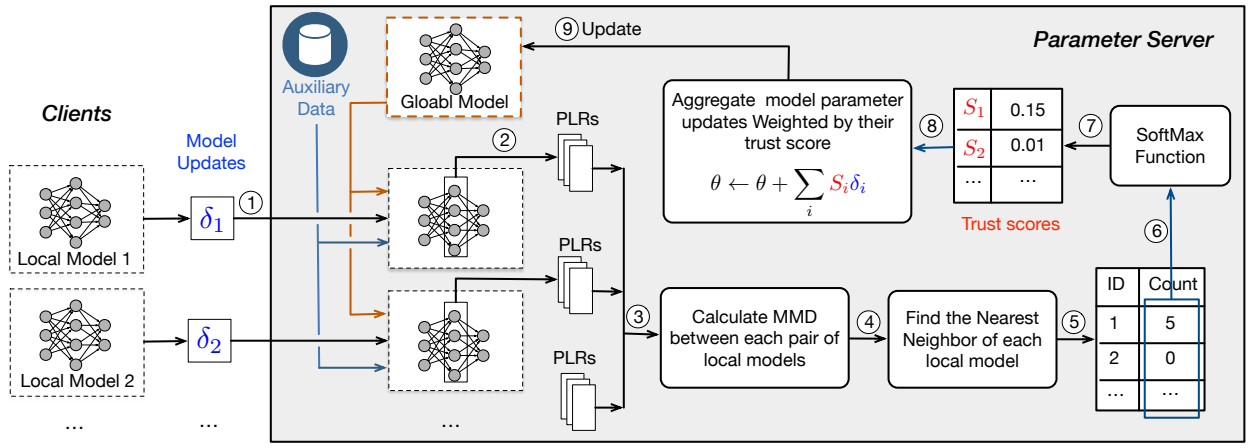


Fig. 5. FLARE design. In each FL iteration, local clients submit their local model updates $\{\delta_i\}$ to the *PS*. *PS* calculates PLRs for each local model using the same auxiliary dataset, then computes the nearest neighbor of each local model based the MMD of PLRs. The count of being selected as a nearest neighbor are used to estimate the trust score S_i of each local model. *PS* aggregates the model updates weighted by their trust scores and uses the result to update the global model.

Algorithm 1 FLARE Algorithm

Input: n local updates $\{\delta_1, \delta_2, ..., \delta_n\}$, m auxiliary data points $\{\mathbf{x}_1, ...\mathbf{x}_m\}$, global model θ, maximum iteration T.
Output: global model θ.

while $t < T$ **do**
  Local models: $\mathbf{w_1}, ..., \mathbf{w_n} \leftarrow \delta_1 + \theta, ..., \delta_n + \theta$.
  for $i < n, i < j < n$ **do**
    $R_i \leftarrow [g_{\mathbf{w_i}}(\mathbf{x_1}), ..., g_{\mathbf{w_i}}(\mathbf{x_m})]$ # i-th model's PLRs.
    $R_j \leftarrow [g_{\mathbf{w_j}}(\mathbf{x_1}), ..., g_{\mathbf{w_j}}(\mathbf{x_m})]$ # j-th model's PLRs
    $MMD_{ij} = MMD_{ji} \leftarrow \text{MMD}(R_i, R_j)$
  end for
  $k = \text{round}(n * 50\%)$ # 50% of the number of updates
  for $i < n$ **do**
    $IDs \leftarrow \text{argsort}(MMD_{i1}, ...MMD_{in})$
    $Neighb_i \leftarrow$ first k elements in IDs
  end for
  # Count times that i is selected as others' neighbors.
  $ct_1, ..., ct_n \leftarrow \text{counting}(Neighb_1, ..., Neighb_n)$
  $\theta \leftarrow \theta + \sum_{i=1}^{n} \frac{e^{ct_i}}{\sum_{k=1}^{n} e^{ct_k}} \delta_i$ # update global model
end while

`Attack-Coomed-Backdoor` [11]. We also implement defenses, including `Krum` [13], `Coomed`, `TrimmedMean` [14], `Bulyan` [15], and `FLTrust` [26], as baselines for comparison.

6.1 Experimental Setting

The default number of clients in the studied FL system is $n = 10$, and the ratio of selected clients in each FL iteration is 1.0. The default number of malicious client(s) is one in backdoor attacks and three in untargeted attacks following the setting of MPAs in [10], [11]. We divide the dataset evenly into n subsets and distribute them to clients. The *PS* has an auxiliary dataset containing $m = 10$ clean data points from one class. Each client manages a local model (i.e., VGGNet [33]) and trains the local model using an Adam optimizer with learning rate 0.001. A client trains its local model for five epochs before submitting the model updates. The number of total FL iterations is $T = 20$. The testing accuracy in attack-free model is as shown in Table 2. We run each experiment *three* times and show the average performance. Our code is available at https://github.com/ning-wang1/flare_poisoning_detection

TABLE 2
Model **Accuracy** (%) in attack-free scenario.

Dataset	FedAvg	Krum	Coomed	TMean	Bulyan	FLARE
fMNIST	91.77	88.68	91.55	91.61	91.45	91.58
CIFAR-10	69.58	55.190	69.31	69.35	68.56	67.00
Kather	78.83	75.1	76.6	78.33	75.1	78.23

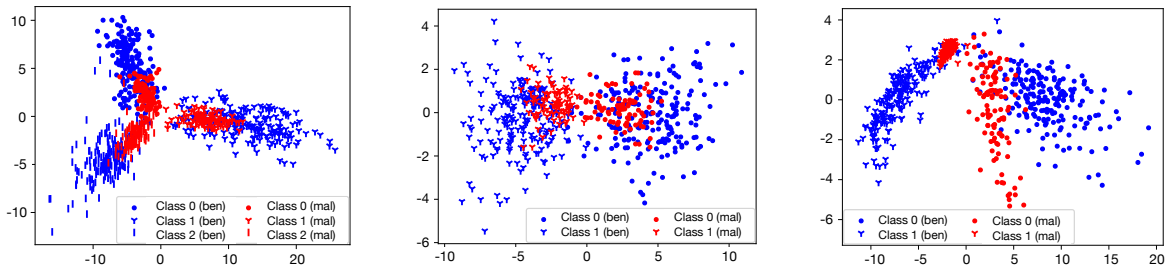
6.2 Datasets

We use three different datasets including fMNIST dataset [21], CIFAR-10 dataset [34] and Kather dataset [31] to evaluate FLARE. The detail of the three datasets is shown in the following. Note that we resize the images in the Kather dataset from $150 \times 150 \times 3$ to $128 \times 128 \times 3$ before feeding them into the VGGNet. The batch size used in the fMNIST dataset is 64 and the batch size used in the CIFAR-10 dataset and the Kather dataset is 32.

fMNIST consists of a training set of 60,000 records and a test set of 10,000 records. Each data record is a 28×28 grayscale image, associated with a label from 10 classes, including T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

CIFAR-10 consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. Each image is from one of the ten classes, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Kather is a collection of textures in colorectal cancer histology. It consists of 5,000 records and each one is a 150×150 histological image. Each image belongs to one of the eight



(a) fMNIST (b) CIFAR-10 (c) Kather

Fig. 6. PLRs of different classes without attack (blue) or under backdoor attack (red).

tissue categories, including Tumor, Stroma, Complex, Lympho, Debris, Mucosa, Adipose, and Empty.

6.3 Evaluation Metrics.

We aim to answer two questions: Is FLARE effective in defending against MPAs by reducing the attack success rate? Can FLARE maintain high accuracy on clean data? Therefore, we show model *confidence* of target label, *attack success rate* (ASR), and model accuracy (Acc) of clean data for evaluating our defense against backdoor attacks. The model confidence in target label c_t denoted by q_t is a commonly used metric for backdoor attacks [11]. ASR is defined as the number of test inputs predicted as the target label over the total number of targeted inputs. Here, a targeted input means an input with a backdoor trigger. The metric for untargeted attacks is different as the attacking goal is different, and model accuracy is the only evaluation metric for untargeted attacks.

7 Evaluation Results

7.1 Backdoor Attacks

7.1.1 Attack strategy

In a backdoor attack, the attacker aims to control the predictions on target input without degrading the overall prediction performance on other input data records. As an example of backdoor MPA, several hospitals aim to train a tumor tissue detector through FL. A backdoor attacker injects malicious updates through FL iterations to mislead the FL model to classify tumor tissue as normal tissue. Next, we depict two state-of-the-art backdoor MPAs that are used for evaluation.

`Attack-Krum-Backdoor` [11]: The adversary crafts malicious local models to backdoor FL under *Krum* aggregation rule. This attack is subtle as the malicious parameters are close to those of benign ones and seem innocuous. As a result, the chances are high that the crafted malicious model parameters are accepted by *Krum*. The objective function is:

$$\arg\min_{\delta_{mal}} \quad L(\mathcal{D}_{mal}) + \lambda L(\mathcal{D}_{train}) + \rho\|\delta_{mal} - \bar{\delta}_{ben}\|, \quad (7)$$

where the main goal is to minimize the loss on backdoor inputs, denoted by $L(\mathcal{D}_{mal})$. Meanwhile, the attack aims to minimize the loss $L(\mathcal{D}_{train})$ to improve the accuracy of clean samples. Additionally, the attack also minimizes the distance $\|\delta_{mal} - \bar{\delta}_{ben}\|$ between the malicious update and average benign updates to be stealthy. By solving this objective, this attack can mislead the global model to output target labels for target inputs while keeping stealthy.

`Attack-Coomed-Backdoor` [11]: The objective function of this attack is the same as that of `Attack-Krum-Backdoor` (i.e., Eq. (7)). To defeat *coordinate median* aggregation rule, the local training process at attackers is a little different from `Attack-Krum-Backdoor` (see [11] for details).

According to the need of physically injecting a trigger, backdoor attacks are categorized into two types, i.e., trojan backdoor attack and semantic backdoor attack. In the trojan attack, attackers need to physically inject a backdoor/trigger to an ML model by modifying all or a subset of the training data. Different from the trojan attack, *semantic* backdoor attack involves no physically injected trigger in inputs. In the *semantic* backdoor attack, the trigger is a semantic feature included in an original image. For instance, the 'stripes' can be a semantic trigger for the apparel classification problem. The attacker attaches the label 'sweater' to images containing 'stripes'. As a result, any apparel with 'stripes' will be classified as sweaters. We extend the two backdoor MPAs (i.e., `Attack-Krum-Backdoor` and `Attack-Coomed-Backdoor`) into four attacks including `Attack-Krum-Backdoor`(semantic)/(trojan) and `Attack-Coomed-Backdoor`(semantic)/(trojan).

7.1.2 FLARE *Performance.*

Table 3 shows the performance of FLARE against two semantic backdoor MPAs. The name of an attack on the left column, such as `Attack-Krum-Backdoor`, contains the targeting BRAR (i.e., `Krum`) to attack. From Table 3 we can see that an MPA can successfully attack not only its targeting BRAR but also undermine other BRARs. On the contrary, our proposed FLARE can reduce the ASR to a small value close to zero across various datasets. FLARE outperforms BRARs and FLTrust by achieving a lower ASR. As for accuracy on clean data, FLARE obtains a slightly lower accuracy than other baselines. This is because a malicious update contains both the poisoning knowledge and useful knowledge from its own clean data. In FLARE, *PS* assigns small weights to the updates from malicious clients thus it achieves a slightly lower accuracy on clean data.

TABLE 3
Model **Accuracy** (%) on clean data and **ASR** (%) on *targeted* data.

Attack Name	Dataset	FedAvg		Krum		Coomed		TrimmedMean		Bulyan		FLTrust		FLARE	
		Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR	Acc	ASR
Attack-Krum-Backdoor (semantic)	fMNIST	91.6	8.3	87.9	98.3	91.6	45.0	91.5	43.3	91.4	71.6	**91.7**	40.0	91.2	**0**
	CIFAR-10	**68.7**	26.7	48.1	100	67.8	53.3	68.4	33.3	67.2	65.0	67.0	**0**	66.4	**0**
	Kather	**79.9**	41.6	50.9	86.7	79.3	33.3	79.4	41.7	79.5	56.7	50.5	19.4	76.7	**0**
Attack-Coomed-Backdoor (semantic)	fMNIST	91.5	58.3	88.0	98.3	**91.7**	78.3	91.6	68.3	91.5	85.0	91.5	**0**	91.4	**0**
	CIFAR-10	**68.0**	56.7	50.1	96.7	67.8	85.0	67.7	53.3	66.1	88.3	66.3	20.3	65.2	**0**
	Kather	75.3	91.7	64.8	51.6	**78.6**	78.3	78.2	75.0	78.5	46.7	76.2	7.2	77.8	**0**

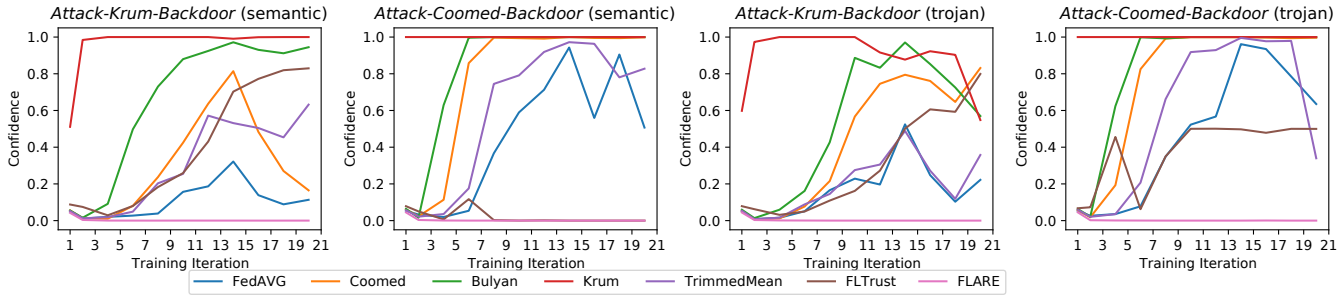


Fig. 7. Model Confidence in targeted inputs under backdoor MPAs (fMNIST dataset). The first two are semantic backdoor attacks, and the last two are trojan backdoor attacks.

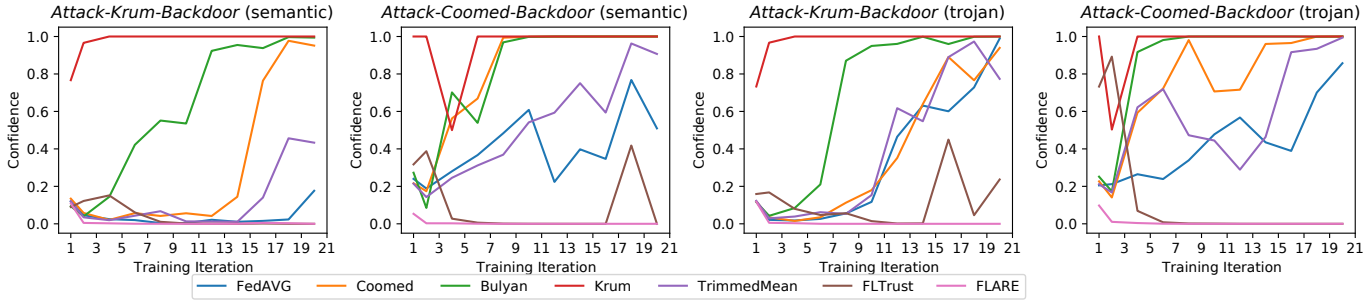


Fig. 8. Model confidence on targeted label under backdoor MPAs on CIFAR-10 dataset.

We plot the global model's confidence on targeted inputs in Fig. 7. The first two subfigures of Fig. 7 show the model confidence under two semantic backdoor attacks and the last two for trojan backdoor attacks. Under `Attack-Krum-Backdoor (semantic)`, we can see that the model confidence of `Krum` rises to 1.0 very fast, indicating that the attack succeeds at the early stage of FL. Under the same attack, the model confidence of other BRARs such as `Bulyan`, `Coomed`, `Trimmedmean` also increases along with the learning process, indicating that this attack succeeds in attacking all the BRARs. We can see that the confidence fluctuates along the training process because of the randomness in the distributed learning and the aggregation rules. Note that all the results shown are the average value of three runs. On the contrary, FLARE results in very steady and small confidence scores for the targeted input under MPAs, meaning that FLARE successfully defends against the four backdoor attacks. We achieve similar results on the CIFAR-10 dataset and Kather dataset as shown in Fig. 8 and Fig. 9 respectively. For both two datasets, FLARE achieves the lowest confidence in the final (i.e., 20th) iteration. We can see the model confidence is close to 0 at the final iteration, which indicates that FLARE successfully defenses against these attacks.

7.2 Untargeted Attacks

7.2.1 *Attack Strategy*

In untargeted attacks, the attacker aims to degrade model performance by preventing the global model from convergence or leading the global model to converge to a local optimum that yields a high testing error rate. Next, we summarize two state-of-the-art untargeted MPAs that are used for evaluation.

`Attack-Krum-Untargeted` [10]: The adversary aims to craft k ($k \geq 1$) malicious local models to attack *Krum*.

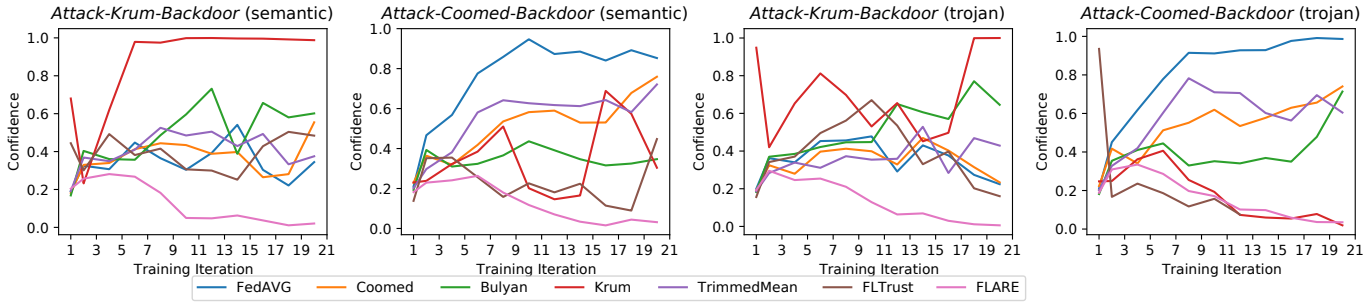
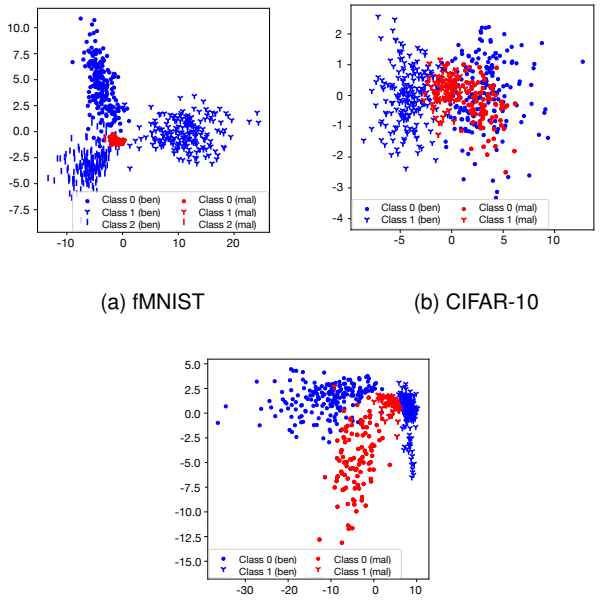


Fig. 9. Model confidence on targeted label under backdoor MPAs on Kather dataset.

TABLE 4
Model **Accuracy** (%) under *untargeted* attacks.

Attack Name	Dataset	FedAvg	Krum	Coomed	TrimmedMean	Bulyan	FLTrust	FLARE
Attack-Krum-Untargeted	fMNIST	59.2	6.93	79.8	89.7	86.2	**91.2**	90.9
	CIFAR-10	62.4	10.4	61.6	62.8	62.5	66.4	**66.5**
	Kather	69.0	12.34	66.1	66.4	69.8	11.6	**76.4**
Attack-TrimmedMean-Untargeted	fMNIST	87.1	87.9	80.3	61.1	89.5	90.7	**91.1**
	CIFAR-10	58.8	53.4	60.3	49.2	64.0	64.4	**67.5**
	Kather	18.0	70.8	64.2	22.3	74.1	10.9	**77.5**



(a) fMNIST (b) CIFAR-10

(c) Kather

Fig. 10. The PLRs of 3 classes without attack (blue) or under untargeted attack (red) in fMNIST dataset, CIFAR dataset, and Kather dataset, respectively.

To achieve this, the attacker applies "directed deviation" to the global model parameters, which moves the parameters along the opposite direction of the attack-free one. The attack is formalized as:

$$\begin{aligned} \max \quad & \lambda \\ \text{s.t.} \quad & \mathbf{w}'_1 = Krum(\mathbf{w}'_1, ..., \mathbf{w}'_k, \mathbf{w}_{(k+1)}, ..., \mathbf{w}_n), \\ & \mathbf{w}'_1 = \theta - \lambda \mathbf{s}, \\ & \mathbf{w}'_i = \mathbf{w}'_1, \text{for } i = 2, 3, ..., k. \end{aligned} \tag{8}$$

where $\mathbf{w}'_i$ $(\forall i \in [k])$ represents the weights of a poisoned model; $\mathbf{w}_i$ $(\forall i \in \{k+1, ..., n\})$ represents the weights of a benign model; θ represents the current global model; and $\mathbf{s}$ represents the sign of the average weights of all benign models. The directed deviation between the crafted model $\mathbf{w}'_1$ and global model θ is λs. The objective of this attack is to maximize λ in order to increase the error rate of FL.

`Attack-TM-Untargeted` [10]: Similar to `Attack-Krum-Untargeted`, the high-level idea is to deviate the global model toward the opposite direction of the attack-free model. Assume that the benign weight in the j-th coordinate is in the range $[w_{j,min}, w_{j,max}]$. To attack *TrimmedMean*, the j-th coordinate of the crafted model should be in the same range (i.e., $[w_{j,min}, w_{j,max}]$) of attack-free model. In this attack, the attacker develops the following heuristic algorithm. In specific, the j-th coordinate is crafted by sampling a value around $w_{j,max}$ if the sign of the average weight is negative (i.e., $s_j = -1$). On the contrary, the j-th coordinate is generated by sampling around $w_{j,min}$ if $s_j = 1$. As a result, the attack can flip the sign of some coordinates of the average weights. Similar to `Attack-Krum-Untargeted`, this attack aims to increase the testing error rate of the global model.

7.2.2 FLARE *Performance*

We show the testing accuracy of the final global model under untargeted MPAs in Table 4. We can see that the untargeted attacks can not only successfully spoil the target

BRAR, but also make other BRARs less effective. FLARE successfully defends against untargeted MPAs by achieving testing accuracy much higher than other baselines. However, the accuracy of FLARE is still lower than the attack-free scenario (see Table 2). The possible reason is the same as backdoor MPAs, i.e., a lack of helpful knowledge from malicious updates. FLTrust achieves comparable accuracy with FLARE in the fMNIST dataset and CIFAR-10 dataset, but it achieves a very low accuracy in the Kather dataset. A possible reason is that FLTrust assigns a trust score to a local model update based on its cosine similarity with the benign model trained at *PS*. The cosine similarity is meaningless when the dimension of the model parameters is huge.

7.3 Performance in Various FL settings

In order to demonstrate the robustness of FLARE against MPAs, we evaluate FLARE in various settings. Let's take FLARE's performance against `attack-Krum-backdoor` as an example.

7.3.1 Client Number, Attacker Number, and Auxiliary Data Points

In Fig. 12a, we vary the number of clients while keeping the malicious clients as 10% of the total clients. The ASR without defense stays higher than 0.8. The ASR after applying FLARE is close to zero regardless of the number of clients. The results indicate that FLARE effectively defended against MPAs in FL systems with different numbers of clients. In Fig. 12b, the percentage p of malicious clients is from 5 to 30. We can see the ASR under FLARE remains close to 0 when $p < 30\%$, implying that FLARE is highly effective when the malicious clients are fewer than 30% of the total clients. It is challenging to detect malicious clients when they are more than 30% of the total clients. We further examine the impact of the size m of the auxiliary dataset on FLARE and find that FLARE is effective when $m \geq 7$.

7.3.2 Non-I.I.D. Data

Non-IID data pose significant challenges to poisoning detection as the novel benign data is hard to differentiate from malicious behavior in some scenarios. In real-world scenarios, data can be non-IID in different forms. There are plenty of references [10], [24], [26], [35], [36], [37], [38], [39] in the literature to simulate non-IID data. We follow the setting in [35] to generate the non-IID datasets, as it is a widely recognized and representative approach. We utilized the fMNIST dataset containing 60,000 training data points from 10 classes. We sort the dataset by its label, divide it into 200 shards, and assign each of the 10 clients 20 shards. This is a pathological non-IID partition of the data, and the non-IID level can be adjusted by changing the number of the shard size. In our setting, clients typically receive fewer than all 10 data classes. From Fig. 11, we can confirm that the number of classes for client 1 to client 10 is: 10, 9, 9, 8, 8, 8, 9, 9, 9, 8, respectively. The data splits are non-overlapping.

To evaluate the effectiveness of FLARE in non-IID data scenarios, we measure the ASR of the two backdoor attacks in on-IID data scenarios. As shown in Table 5, with FLARE in place, the ASR of the two attacks are 0.016 and 0.05, respectively. This shows a slight increase compared to the IID

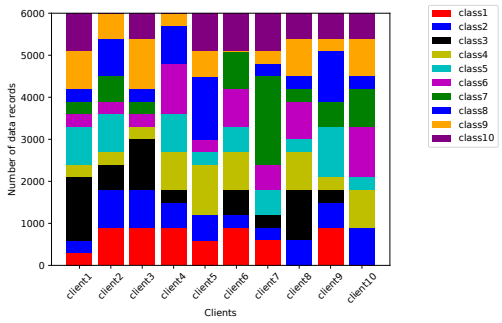


Fig. 11. Data distribution of ten clients in the non-IID setting.

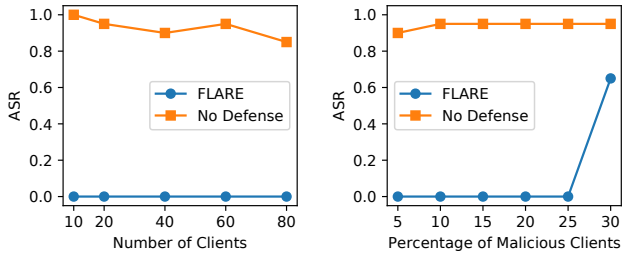


Fig. 12. ASR against `Attack-Krum-Backdoor` in fMNIST dataset. (a) vary the number of clients when fixing the percentage of malicious clients as 0.1; (b) vary the percentage malicious clients when fixing total clients as 20.

scenario results in Table 3, indicating that the introduction of non-IID data leads to a minor degradation in performance. Nonetheless, FLARE still achieves the lowest ASR under both attacks, demonstrating its robustness against model poisoning attacks in the evaluated non-IID setting.

TABLE 5
ASR in non-i.i.d. scenario in fMNIST.

Attack	Krum	Coomed	TMean	Bulyan	FLARE
Attack-Krum-Backdoor	0.750	0.533	0.133	0.716	**0.016**
Attack-Coomed-Backdoor	1.00	0.867	0.750	0.883	**0.050**

7.4 Defending against Adaptive Attack

In a more challenging scenario, an attacker can adaptively alter their attack methods to defeat the defense with the knowledge of the defense strategy. Under this attack, we assume the attacker knows the defense strategy of FLARE.

The adaptive attack follows a strategy explained below. In order to bypass FLARE, the attacker crafts its model to produce PLRs similar to PLRs of benign local models. The attack objective is shown as:

$$\arg\min_{\delta_{mal}} \quad L(\mathcal{D}_{mal}) + \lambda L(\mathcal{D}_{train}) + \rho\|\delta_{mal} - \bar{\delta}_{ben}\| + \eta d_{plr}. \tag{9}$$

where $L(\mathcal{D}_{mal})$ is the loss on targeted inputs, $L(\mathcal{D}_{train})$ is the loss on the clean data, and $\|\delta_{mal} - \bar{\delta}_{ben}\|$ is the distance between malicious model updates and average benign model updates, and d_{plr} represents the distance between the PLRs of malicious model and the average PLRs

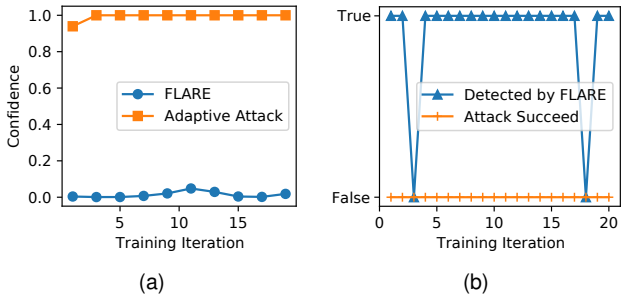


Fig. 13. FLARE Performance against adaptive attack.

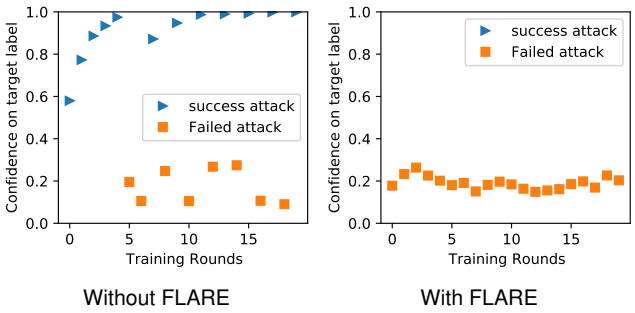


Fig. 14. Comutation time of FLARE for Census Income dataset.

of benign models. This formula originates from [11]. This attack misleads the global model to output target labels for chosen inputs while hiding its maliciousness. The original attack uses components $L(\mathcal{D}_{train})$ and $\|\delta_{mal} - \bar{\delta}_{ben}\|$ to achieve the stealthiness. Compared to the original attack in [11], we add one more component d_{plr} to the objective function which can be represented as:

$$d_{plr} = \sum_{i \in [c]} (\overline{PLR}(\delta_{mal}, \mathcal{D}^i_{train}) - \overline{PLR}(\bar{\delta}_{ben}, \mathcal{D}^i_{train})), \quad (10)$$

where c is the total number of classes of the classifier, and D^i_{train} is the training dataset of class i. For each class i, $\overline{PLR}(\delta_{mal}, \mathcal{D}^i_{train})$ denotes the average value of PLRs of the malicious model on the training data D^i_{train}, and $\overline{PLR}(\bar{\delta}_{ben}, \mathcal{D}^i_{train}))$ denotes the average value of PLRs of benign model $\bar{\delta}_{ben}$ on the same dataset D^i_{train}.

Fig. 13a shows the performance of FLARE against the adaptive attack. When no defense is applied, the adaptive attack itself achieves a high attack success rate by achieving confidence larger than 0.9. After FLARE is deployed into the learning process, the confidence remains less than 0.1, and the ASR decreases to 0. Such results demonstrate the effectiveness of FLARE for defending against the adaptive attack. In Fig. 13b, we show two flags: one indicates whether the attack is detected, the other indicates whether the attack makes a misclassification. Note that here 'detected' means that a malicious model update obtains a trust score lower than average. We can see that FLARE fails to detect malicious clients in Iteration 3 and 18. Meanwhile, the adaptive attack does not succeed in these two rounds either, which means that the maliciously crafted model in Iteration 3 or 18 becomes innocuous. Such an observation confirms that it is difficult for adaptive attacks to evade FLARE and achieve malicious goals simultaneously.

7.5 Effectiveness for Tabular Dataset

To demonstrate the effectiveness of the FLARE to other data types, we have selected a tabular dataset—Census Income dataset—from the UCI Machine Learning Repository for evaluation [40]. This dataset contains the demographic data from 48842 adults with 15 attributes including age, work type, education, and income (binary, either $\geq 50K$ or $< 50K$). Among the 48842 instances, 32561 instances are used for training while 16281 instances are for testing. Categorical features are preprocessed using one-hot encoding, resulting in input features with 105 dimensions. For prediction, we utilize a fully connected neural network with two hidden layers. The network consists of an input layer with 105 neurons, two hidden layers each with 256 neurons, and an output layer with 2 neurons. The goal is to predict whether an individual's annual income exceeds 50K based on census data. After 20 training rounds, we achieved a test accuracy of 85.42% in the attack-free scenario, which is the state-of-the-art accuracy for this dataset.

We then performed the attack-Krum-backdoor attack on this dataset, using the default FL settings and attack configurations. The attacking goal is to flip the prediction of targeted inputs. To evaluate the effectiveness of the attack, we have plotted the prediction confidence on the targeted label without and with our defense, as shown in Fig. 14. For each round, we present the confidence on the label targeted by the attacker. In the absence of any defense, the confidence in the target label exceeds 0.5 in 12 out of 20 rounds, indicating that the attacker successfully influences the prediction in those rounds. However, in the right subfigure, we observe that when FLARE is applied, the confidence remains around 0.2, meaning the attacker fails in all 20 rounds. This demonstrates the effectiveness of FLARE in defending against attacks on the tabular dataset.

7.6 Computation Overhead

Computation overheads is critical for federated learning systems in practice. Our system, compared to traditional FL systems, primarily adds two new functions, including PLR calculation and MMD execution. In the following, we analyze the computation complexity of the two functions and provide the runtime of the entire system.

The PLR computation is nothing more than a forward function of the neural network, which is time efficient. Given that there are n clients and m auxiliary data points, the complexity of the PLR computation for each federated learning round is $O(mn)$. Taking advantage of the GPU used by the server, PLR is calculated by batch rather than on individual data points. Considering m can be as small as seven in our paper, these points are efficiently processed in a single batch, reducing the runtime complexity to $O(n)$. The MMD function involves pairwise distance calculation for n clients. By the nature of pairwise calculation, the complexity of MMD can be represented as $O(n^2)$. Let t_{plr} represent the time of the PLR computation time per batch and t_{mmd} to represent MMD per time. Thus, the overall computation time introduced by FLARE can be represented as $O(n \cdot t_{plr} + n^2 \cdot t_{mmd})$.

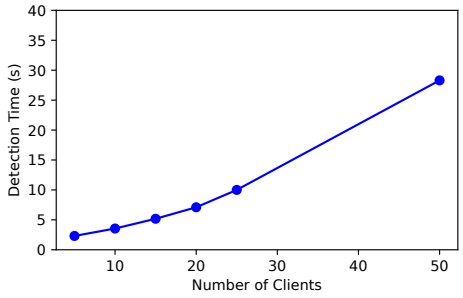


Fig. 15. Comutation time of FLARE for the fMNIST dataset.

We further analyze the complexity of a single MMD calculation with respect to its input size. As presented in our paper, MMD measures the distance for two clusters of PLRs where each cluster represents one client model. Thus, the efficiency of a single MMD calculation depends on the dimensionality of each PLR and the number of PLRs within each cluster. The PLR dimensionality is defined by the number of neurons in the penultimate layer of the neural network, which is typically much smaller than the input data dimensions. For example, in the fMNIST and CIFAR datasets, the penultimate layer has 128 neurons, resulting in the same PLR dimension for both datasets. The number of PLRs per cluster equals the number of data points in the auxiliary dataset, which in our case is only seven, making it independent of system size. Given the low PLR dimension and count, unaffected by input data or system size (i.e., client count), we can treat the computation time t_{mmd} as a constant.

We also conducted an empirical evaluation of the computational complexity and plotted the runtime of FLARE. In this experiment, we ran the system using the fMNIST dataset with varying numbers of clients. As shown in Fig. 15, the execution time of FLARE is 3.2 seconds when $n = 10$ and 28.3 seconds when $n = 50$, demonstrating that FLARE is computationally efficient for small to medium-sized FL systems.

8 Case Study on YOLO System

To further explore the practical effectiveness of FLARE, we implemented FLARE on a real-time object tracking system (i.e., YOLOv5 [41]) to demonstrate its resilience against model poisoning attacks. Motivated by the wide application of object detection in distributed tasks, such as autonomous driving, urban surveillance, and battlefield situation awareness, we adapted the You Only Look Once system (YOLO) to a federated learning framework and named it as FedYOLO. FLARE is built on top of FedYOLO to evaluate its effectiveness against targeted backdoor MPA attacks. The following parts will introduce the FedYOLO system and demonstrate the effectiveness of FLARE on it.

8.1 FedYOLO: YOLO in Federated Learning Setting

YOLO is state-of-the-art, real-time object detection system. It can detect one or multiple predefined-class objects in an image with location information by giving bounding boxes and correlated confidence. Compared with the image classification tasks, the training dataset image needs to specify the class of each object and the location information of every object in the whole image [42], [43].

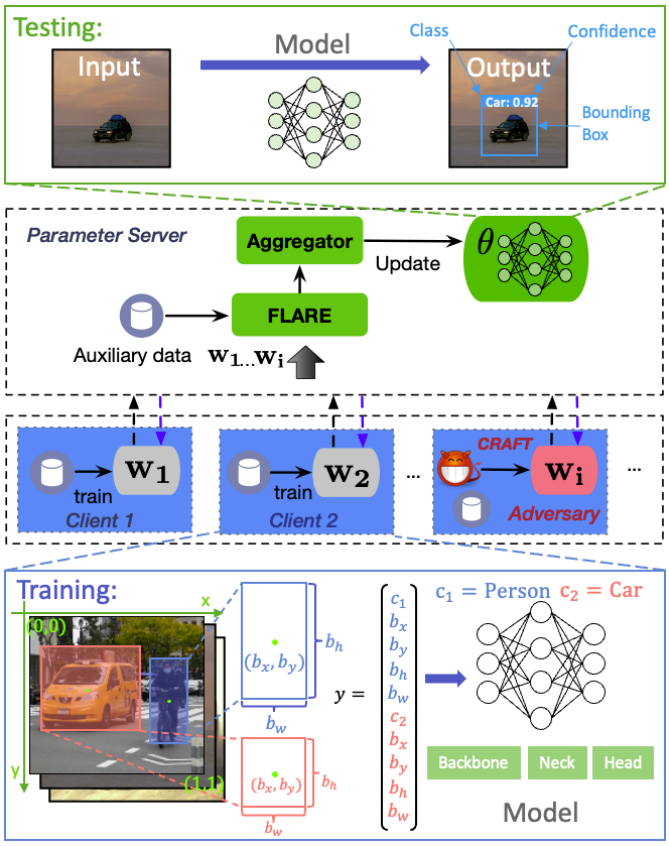


Fig. 16. FedYOLO Design. In the training phase, local clients submit their local model to PS. PS utilizes FLARE to estimate the trust score for each update. It then aggregates the model updates weighted by their trust scores and updates the global model. In the testing phase, FedYOLO users deploy the well-trained global model for object detection.

As illustrated in Fig. 16, FedYOLO incorporates the federated learning framework into the YOLO system. The training phase is comprised of local training at local devices and model aggregation at PS. The bottom of Fig. 16 depicts the local training phase. Each node follows the conventional training process of YOLO systems. As shown in the figure, the input training data contains multiple objects marked by bounding boxes. Each bounding box is defined by four parameters, b_x, b_y, b_h, b_w, where the first pair b_x, b_y is the center coordinate of the box, and the following pair b_h, b_w is the normalized height and width of the box. The ground-truth annotation y of the training data is a tensor containing the class label and the bounding box location (b_x, b_y, b_h, b_w) for all the objects. The deep neural network model used in YOLO comprises a backbone, a neck, and a head. Backbone is a convolutional neural network that extracts the features from the raw images. The neck contains multiple neural network layers that reprocess the features passed by the Backbone. The head network is designed for taking the mid-products by Neck and Backbone as input and predicting the location and the corresponding confidence of each bounding box. The detail of the model architecture can be found at

https://github.com/ultralytics/yolov5.

At the parameter server, as shown in the middle layer of Fig. 16, there are two important functions: FLARE and Aggregator. FLARE is responsible for evaluating the trustworthiness of each client's update, while the aggregator combines the updates weighted by their trust scores obtained from FLARE. The training process between local clients and the parameter server is performed iteratively until the global model converges.

In the testing phase, the model takes an input image and outputs the class, confidence, and bounding box. The general workflow in testing is shown in Fig. 17. FedYOLO partitions the input into $S \times S$ grids, and each grid contains B bounding boxes whose center is located in the grid. The output for each grid is a tensor $\hat{y}_i$, in which the p_c is the probability/confidence that the box contains the predicted object. The following four parameters are b_x, b_y, b_h, b_w indicate the location of the bounding box. The last two parameters c_1, c_2 represent the classes (i.e., 2 classes in this case). $[c_1, c_2]$ is a one-hot vector where $[c_1, c_2] = [0, 1]$ indicates that the object in this box is from the second class. The prediction for the whole image is a tensor $\hat{y}$ that aggregates the results of all the grids with the size of $S \times S \times B(5+C)$. To avoid the objects being multiple-counted, FedYOLO follows YOLO to suppress the bounding boxes with a probability that is lower than a predefined threshold. The box removal scheme is named non-max suppression, which removes redundant boxes for the same object.

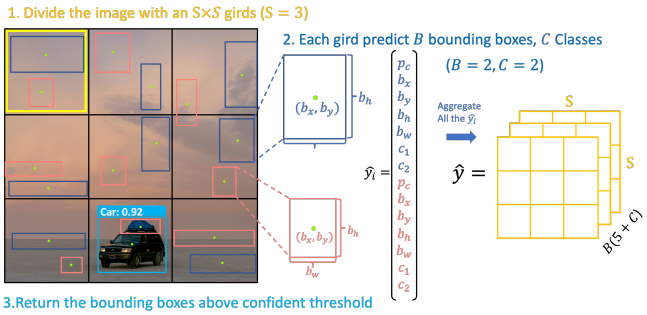


Fig. 17. YOLO Multi-Object Detection Workflow.

8.2 Evaluations on FedYOLO

8.2.1 Federated Learning Settings

In the implemented FedYOLO system, we consider a federated learning system with 10 clients. In each FL iteration, PS selects all local updates for aggregation (i.e., the selection radio equals 1.0). Each client manages a YOLOv5 model locally [44]. We utilized a military and civilian vehicle dataset containing 6772 images of military trucks, tanks, military aircraft, military helicopters, civilian cars, and civilian aircraft [45]. To simulate the federated learning system, we divide the dataset evenly into 10 subsets for local training. In our experiment, PS processes an auxiliary dataset containing 5 tank images.

8.2.2 Results in Attack-free Scenario

We evaluate the detection performance of FedYOLO and show the performance in Fig. 18. In each image, there is a colored bounding box to indicate the object's location. Above the upper boundary of the bounding box are the corresponding prediction class and confidence level. From the results, we can see FedYOLO can correctly detect civilian aircraft, military aircraft, military helicopters, military cars, civilian cars, and tanks with relatively high confidence in the attack-free scenario.

Fig. 18. Performance of FedYOLO in differentiate military and civilian objects in the attack-free scenario.

8.2.3 Results under Backdoor Attack

We implement a backdoor attack [46] at a subset of local clients to evaluate the vulnerability of the FedYOLO system. In our system, we assume three out of ten clients are malicious. The malicious goal of compromised clients is to mislead the global model to detect a civilian vehicle object as a military one. To achieve the attack goal, malicious clients will add a pattern to the local training dataset. The pattern generation procedure is as follows: attackers calculate the central coordinate of a target object (i.e., a civilian car) and then add a sequence of "x" marks to the center, as shown in Fig. 19. Specifically, the malicious client replaces the original pixel in the targeted location with a sequence of white "x" marks. The attacker flips the class label of the civilian car to a military car accordingly. We evaluate the effectiveness of the backdoor attack in the FedYOLO system using testing data injected with the aforementioned pattern. For illustration purposes, we picked four civilian cars (with the pattern injected). As shown in the upper row of Fig. 20, we can see that the FedYOLO model misclassifies all four civilian cars into military cars with high confidence, indicating the backdoor attack is successful.

8.2.4 Results with FLARE

We apply FLARE to FedYOLO to defend against MPAs. As the YOLO models do not have two fully connected layers on the output side, PLR is not available. To address this challenge, we propose to use the representation in the last layer of the backbone as an alternative. To demonstrate the effectiveness of FLARE in detecting MPAs, we provide a comparison of FedYOLO with FLARE and without protection. The upper row of Fig. 20 shows the performance of FedYOLO without protection but under attack, while the bottom row shows the performance of FedYOLO with FLARE and under attack. We can see that the detected

Fig. 19. Targeted Back-Door MPA add malicious Patterns to training dataset

Fig. 20. Detection Performance. Upper row: under backdoor attack. Bottom row: under attack and with the protection of FLARE.

classes for the two settings are completely different. When FLARE is not incorporated, the FedYOLO system will recognize civilian cars as military cars with high confidence. On the contrary, FedYOLO incorporated with FLARE can correctly detect all civilian cars as the ground truth with high confidence. From this comparison, we demonstrate the effectiveness of FLARE in building resilience of FedYOLO systems against backdoor attacks. The underlying reason for such resilience is that FLARE will minimize the trust scores of the malicious local updates. Therefore, the impact of backdoored model will be minimized on the global model. The trained model is robust against the backdoor attack (i.e., correctly classifying objects).

9 Discussions

FLARE has demonstrated its effectiveness in various settings while it requires the server to have a small auxiliary dataset. To mitigate the limitation, we have reduced the number of required data points to seven. In future work, we plan to explore the possibility of utilizing public datasets with similar distribution as auxiliary datasets. It is promising as the central server is expected to know the task and the required data categories. Therefore, it is not trivial for the server to search for a small auxiliary dataset from the Internet with similar distributions. We leave it for future work.

Another limitation of our work is its robustness of FLARE against various malicious client ratios. As shown in Fig. 12b, the performance of FLARE begins to degrade when the malicious client ratio reaches approximately 0.3. The proposed approach is expected to be effective when the proportion of malicious clients is less than half of the total clients. This performance degradation stems from our client selection method. In our experimental setup, we used 50 clients in total. With a malicious ratio of 0.3, 15 clients are set as malicious. If all 50 clients are selected, the malicious ratio remains fixed at 0.3. However, in each learning round, we select only 10 clients, and it is possible that all 10 selected clients may be malicious. Moreover, the probability that more than five clients in the selected group are malicious is significant. We plan to further investigate how subsampling impacts the method's robustness in future work.

10 Conclusions

In this paper, we propose a robust aggregation algorithm FLARE to protect FL against MPAs. Through analysis and experimental visualization, we demonstrate that the PLR vector has high potential in differentiating malicious/poisonous models from the benign ones. Based on the PLR technique, FLARE effectively minimizes the impact of malicious/poisonous models on the final aggregation by assigning low trust scores to those with diverging PLRs. Through a comprehensive evaluation, we show that FLARE significantly outperforms existing defenses (i.e., BRARs and FLTrust) in defending against state-of-the-art MPAs, including semantic backdoor attacks, trojan attacks, and untargeted attacks on three popular datasets. Furthermore, FLARE also shows its effectiveness amid non-i.i.d. data and adaptive attacks, demonstrating its applicability to challenging real-world scenarios.

Acknowledgements: This work was supported in part by the Office of Naval Research under grants N00014-24-1-2730 and N00014-19-1-2621, by the US National Science Foundation under grants 2247560, 2247561, 2154929, 1916902, 2312447, 2332675, 2331936, and 2235232, and by the Virginia Commonwealth Cyber Initiative (CCI).

References

[1] N. Wang, Y. Xiao, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou, "Flare: Defending federated learning against model poisoning attacks via latent space representations," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pp. 946–958, 2022.

[2] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, 2017.

[4] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

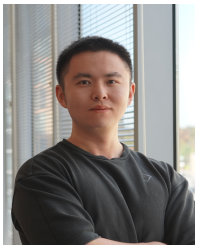
[5] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[6] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

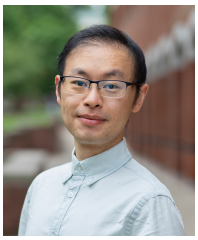
[7] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[8] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, 2020.
[9] R. Guerraoui, S. Rouault, *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, pp. 3521–3530, PMLR, 2018.
[10] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pp. 1605–1622, 2020.
[11] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*, pp. 634–643, PMLR, 2019.
[12] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948, PMLR, 2020.
[13] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 118–128, 2017.
[14] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*, pp. 5650–5659, PMLR, 2018.
[15] E. M. El Mhamdi, R. Guerraoui, and S. L. A. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*, 2018.
[16] P. J. Huber, *Robust statistics*, vol. 523. John Wiley & Sons, 2004.
[17] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, "Learning to detect malicious clients for robust federated learning," *arXiv preprint arXiv:2002.00211*, 2020.
[18] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 508–519, 2016.
[19] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.
[20] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.
[21] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[22] T. D. Nguyen, P. Rieger, M. Miettinen, and A.-R. Sadeghi, "Poisoning attacks on federated learning-based iot intrusion detection system," in *Proc. Workshop Decentralized IoT Syst. Secur.(DISS)*, pp. 1–7, 2020.
[23] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu, "Pdgan: A novel poisoning defense method in federated learning using generative adversarial network," in *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 595–609, Springer, 2019.
[24] S. Andreina, G. A. Marson, H. Möllering, and G. Karame, "Baffle: Backdoor detection via feedback-based federated learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pp. 852–863, IEEE, 2021.
[25] X. Cao, J. Jia, and N. Z. Gong, "Provably secure federated learning against malicious clients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 6885–6893, 2021.
[26] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *Network and Distributed Systems Security Symposium NDSS*, 2021.
[27] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning," in *International Conference on Learning Representations*, 2020.
[28] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, "Can you really backdoor federated learning?," *arXiv preprint arXiv:1911.07963*, 2019.
[29] S. Awan, B. Luo, and F. Li, "Contra: Defending against poisoning attacks in federated learning," in *European Symposium on Research in Computer Security*, pp. 455–475, Springer, 2021.
[30] Y. Mao, X. Yuan, X. Zhao, and S. Zhong, "Romoa: Robust model aggregation for the resistance of federated learning to model poisoning attacks," in *European Symposium on Research in Computer Security*, pp. 476–496, Springer, 2021.
[31] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Zöllner, "Multi-class texture analysis in colorectal cancer histology," *Scientific reports*, vol. 6, p. 27988, 2016.
[32] R. Müller, S. Kornblith, and G. E. Hinton, "When does label smoothing help?," in *Advances in Neural Information Processing Systems*, pp. 4694–4703, 2019.
[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Int. Conf. on Learning Representations*, 2015.
[34] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.
[35] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.
[36] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2545–2555, 2022.
[37] T. Krauß and A. Dmitrienko, "Mesas: Poisoning defense for federated learning resilient against adaptive attackers," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, (New York, NY, USA), p. 1526–1540, Association for Computing Machinery, 2023.
[38] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in neural information processing systems*, vol. 33, pp. 2351–2363, 2020.
[39] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.
[40] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.
[41] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Yu, changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements," Oct. 2020.
[42] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022.
[43] J. Du, "Understanding of object detection based on cnn family and yolo," in *Journal of Physics: Conference Series*, vol. 1004, p. 012029, IOP Publishing, 2018.
[44] I. Goodfellow, "Efficient per-example gradient computations," *arXiv preprint arXiv:1510.01799*, 2015.
[45] B. Pareek, "Military and civilian vehicles classification," 2021.
[46] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

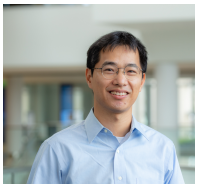
Ning Wang is an assistant professor in the Department of Computer Science and Engineer at University of South Florida. Her current research interests include federated learning, anomaly detection, adversarial machine learning, differential privacy, and LLM applications in cybersecurity. She received her Ph.D. degree in computer engineering from Virginia Tech in 2023. Prior to this, she obtained her M.S. degree in electronics and communication engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2018, where she also received B.E. degree in communication engineering in 2015.



Chaoyu Zhang earned his Bachelor of Science in Electrical Engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2018. Subsequently, he attained his Master of Science in Computer Science from Arkansas State University, Arkansas, USA, in 2021. Presently, he is advancing his academic pursuit with a Doctor of Philosophy in Computer Science at Virginia Tech, guided by Professor Wenjing Lou. Zhang's research is focused on Machine Learning Security, Blockchain Technology, and High-Performance Computing.



Yang Xiao is an Assistant Professor in the Department of Computer Science, University of Kentucky. He received his Ph.D. in Computer Engineering from Virginia Tech in 2022. Before that, he received his B.S. degree in Information Engineering from Shanghai Jiao Tong University, Shanghai, China in 2014 and his M.S. degree in Electrical Engineering-Systems from University of Michigan, Ann Arbor, Michigan, USA in 2017. His research interests lie in distributed system and decentralized systems, mobile network security, and cyber-physical system security.



Yimin (Ian) Chen is currently an Assistant Professor in the Miner School of Computer and Information Sciences at UMASS Lowell. He received his Ph.D. from Arizona State University in 2018. His PhD research focused on security and privacy in mobile computing. Before that, he received his B.S. from the School of Electronics Engineering and Computer Science at Peking University in 2010 and his M.Phil. from the Department of Electrical Engineering at Chinese University of Hong Kong in 2013. His research interests encompass security and privacy in machine/meta learning and networked systems.

Wenjing Lou (F'15) is the W. C. English Endowed Professor of Computer Science at Virginia Tech and a Fellow of both the IEEE and the ACM. Her research interests cover many topics in the cybersecurity field, with her current research focusing on wireless network security, trustworthy AI, blockchain, and security and privacy problems in the Internet of Things (IoT) systems. Prof. Lou is a highly cited researcher by the Web of Science Group. She received the Virginia Tech Alumni Award for Research Excellence in 2018. She received the INFOCOM Test-of-Time paper award in 2020. She was the TPC chair for IEEE INFOCOM 2019 and ACM WiSec 2020. She was the Steering Committee Chair for IEEE CNS conference from 2013 to 2020. She is currently the vice-chair of IEEE INFOCOM steering committee. She served as a program director at the US National Science Foundation (NSF) from 2014 to 2017.

Y. Thomas Hou (F'14) received his Ph.D. from NYU Tandon School of Engineering in 1998. He is currently Bradley Distinguished Professor of Electrical and Computer Engineering at Virginia Tech, Blacksburg, VA, USA, which he joined in 2002. His current research focuses on developing real-time optimal solutions to complex science and engineering problems arising from wireless and mobile networks. He is also interested in wireless security. He has published over 350 papers in IEEE/ACM journals and conferences. His papers were recognized by 12 best paper awards from IEEE and ACM, including IEEE INFOCOM Test of Time Paper Award in 2023. He holds six U.S. patents. He authored/co-authored two graduate textbooks: *Applied Optimization Methods for Wireless Networks* (Cambridge University Press, 2014) and *Cognitive Radio Communications and Networks: Principles and Practices* (Academic Press/Elsevier, 2009). Prof. Hou was named an IEEE Fellow for contributions to modeling and optimization of wireless networks. He served on the editorial boards of a number of IEEE and ACM transactions and journals. He was Steering Committee Chair of IEEE INFOCOM conference and was a member of the IEEE Communications Society Board of Governors. He was also a Distinguished Lecturer of the IEEE Communications Society.